Boosting

Nishant Mehta

Lecture 4

“How is it that a committee of blockheads can somehow arrive at highly reasoned decisions, despite the weak judgment of the individual members? How can the shaky separate views of a panel of dolts be combined into a single opinion that is very likely to be correct?

… this unlikely strategy turns out to form the basis of *boosting*

… at its core, boosting solves hard machine-learning problems by forming a very smart committee of grossly incompetent but carefully selected members.”

—Schapire and Freund (Boosting: Foundations and Algorithms, 2012)

Weak Learners

Typically easy to find rules of thumb (simple rules) that just perform better than chance (e.g., error at most 49%)

Example: If “bitcoin” appears in an email, predict spam

Typically hard to find a single rule that is highly accurate (error close to 0%)

Weak Learners

A ***weak learner*** is a learning algorithm that produces a simple, "better than chance" rule (i.e. a rule of thumb).

Examples:

- Decision stumps (or low-depth decision trees)
- Linear predictors (or nonlinear predictors using few features)

A decision stump-learning algorithm can probably learn a decision stump that performs better than chance.

The hope: Can we "boost" the performance of decision stumps by training different stumps that focus on different parts of the data, and then combining them?

Recall: Random forests using decision stumps was one way of combining predictions of weak learners. Random forests are an ensemble method that combines rules of thumb. But the stumps aren't very specialized and the method of combination is very naive (simple majority vote).

Generic Boosting Algorithm

Given: A weak learning algorithm and a training set

Initially, give all examples in training set the same weight

Run weak learner on weighted version of training set, producing a *weak hypothesis*

Change emphasis/weights on the examples somehow

Repeat T times

Form a final predictor that somehow combines the predictions of the learned weak hypotheses

Key Questions

How to reweight examples in each round?

How to combine weak hypotheses into a final predictor?

Key Questions

How to reweight examples in each round?

Increase weight on examples that were misclassified by the most recent weak hypothesis

How to combine weak hypotheses into a final predictor?

Use a weighted majority predictor where the voters are the learned weak hypotheses
(and give more weight to more accurate rules)

Boosting

The setting is binary classification, and we assume the data is labeled by some concept c in a known concept class $\mathcal{C}$.

a classifier $c\colon \mathcal{X} \to \{-1, +1\}$

a set of classifiers $\mathcal{C} \subseteq \mathcal{X}^{\{-1,+1\}}$

A *weak learner* is a learning algorithm that, no matter the distribution over the inputs (the x's), returns a hypothesis that performs strictly better than chance (e.g., error at most 49%).

A *strong learner* is a learning algorithm that can get error at most ε by using a number of examples that grows at most polynomially in $1/\varepsilon$.

A *boosting algorithm* is an algorithm that "boosts" a weak learner into a strong learner: by making repeated calls to the weak learner and by using enough data, a boosting algorithm can provably return a hypothesis of arbitrarily low error (at most ε).

"error" means "true error" = "risk"

Weak and Strong Learnability, Formally

Weak Learner $\mathcal{A}_{\text{weak}}$

For fixed $\delta_0 > 0$ and $\gamma > 0$, for all distributions over the inputs, given a constant number of examples, with probability at least δ_0, $\mathcal{A}_{\text{weak}}$ returns a hypothesis with error at most $\frac{1}{2} - \gamma$.

Strong Learner $\mathcal{A}_{\text{strong}}$

For all $\delta \in (0, 1)$ and $\varepsilon > 0$, for all distributions over the inputs, given polynomially (in $\frac{1}{\varepsilon}$ and $\frac{1}{\delta}$) many examples, with probability at least $1 - \delta$, $\mathcal{A}_{\text{strong}}$ returns a hypothesis with error at most ε.

If there exists a weak learner for some concept class $\mathcal{C}$, then we say that weak learnability holds.

If there exists a strong learner for some concept class $\mathcal{C}$, then we say that strong learnability holds.

Michael Kearns and Leslie Valiant (1988):

“Is weak learnability equivalent to strong learnability?”

Michael Kearns and Leslie Valiant (1988):

> “Is weak learnability equivalent to strong learnability?”

Answer: “Yes” —Robert Schapire (1989)

A Boosting Algorithm

Given: Labeled training set $(x_1, y_1), \ldots, (x_n, y_n)$ with labels $y_i \in \{-1, +1\}$

Weak learner

for $t = 1, 2, \ldots, T$:

Choose distribution D_t over $\{1, \ldots, n\}$ $\quad \varepsilon_t = \Pr_{i \sim D_t}(h_t(x_i) \neq y_i)$

Using weak learner, find weak hypothesis h_t with small error ε_t under distribution D_t

Output final classifier H that combines predictions of $h_1, \ldots, h_T$

Missing details: Precisely how to set the weights $D_t(1), \ldots, D_t(n)$?
How to form the final classifier H?

AdaBoost

Set $D_1(i) = 1/n$ for all i

for $t = 1, 2, \ldots, T$:

Run weak learner on training set weighted by D_t, giving h_t

Compute error of h_t under D_t: $\varepsilon_t = \Pr_{i \sim D_t}(h_t(x_i) \neq y_i)$

Update distribution: $D_{t+1}(i) = \dfrac{D_t(i)}{Z_t} \cdot \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$

normalization constant

Magic number!

$$\alpha_t = \frac{1}{2} \log \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$

Note: $\alpha_t > 0$ since weak learner should do better than chance (should have $\varepsilon_t < 1/2$)

AdaBoost

Set $D_1(i) = 1/n$ for all i

for $t = 1, 2, \ldots, T$:

Run weak learner on training set weighted by D_t, giving h_t

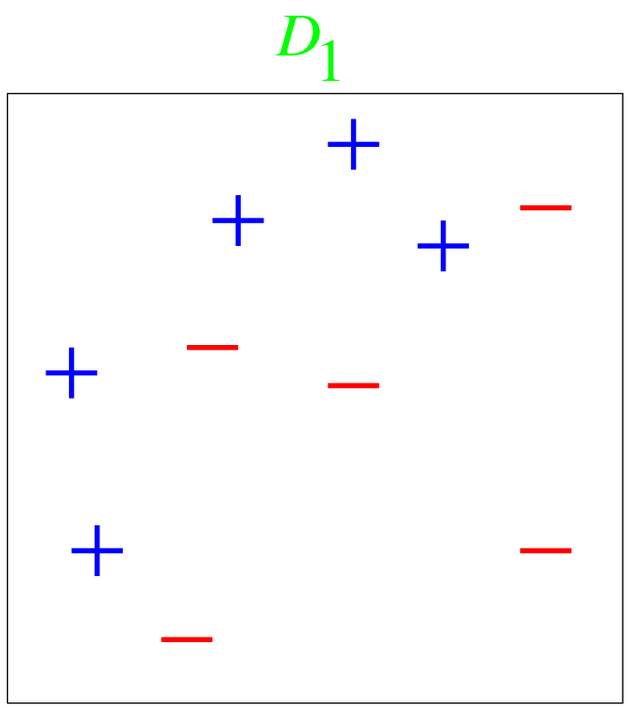
Compute error of h_t under D_t: $\varepsilon_t = \Pr_{i \sim D_t}(h_t(x_i) \neq y_i)$

Update distribution: $D_{t+1}(i) = \dfrac{D_t(i)}{Z_t} \cdot e^{-\alpha_t y_i h_t(x_i)}$

Output final classifier: $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

$$\alpha_t = \frac{1}{2} \log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) > 0$$

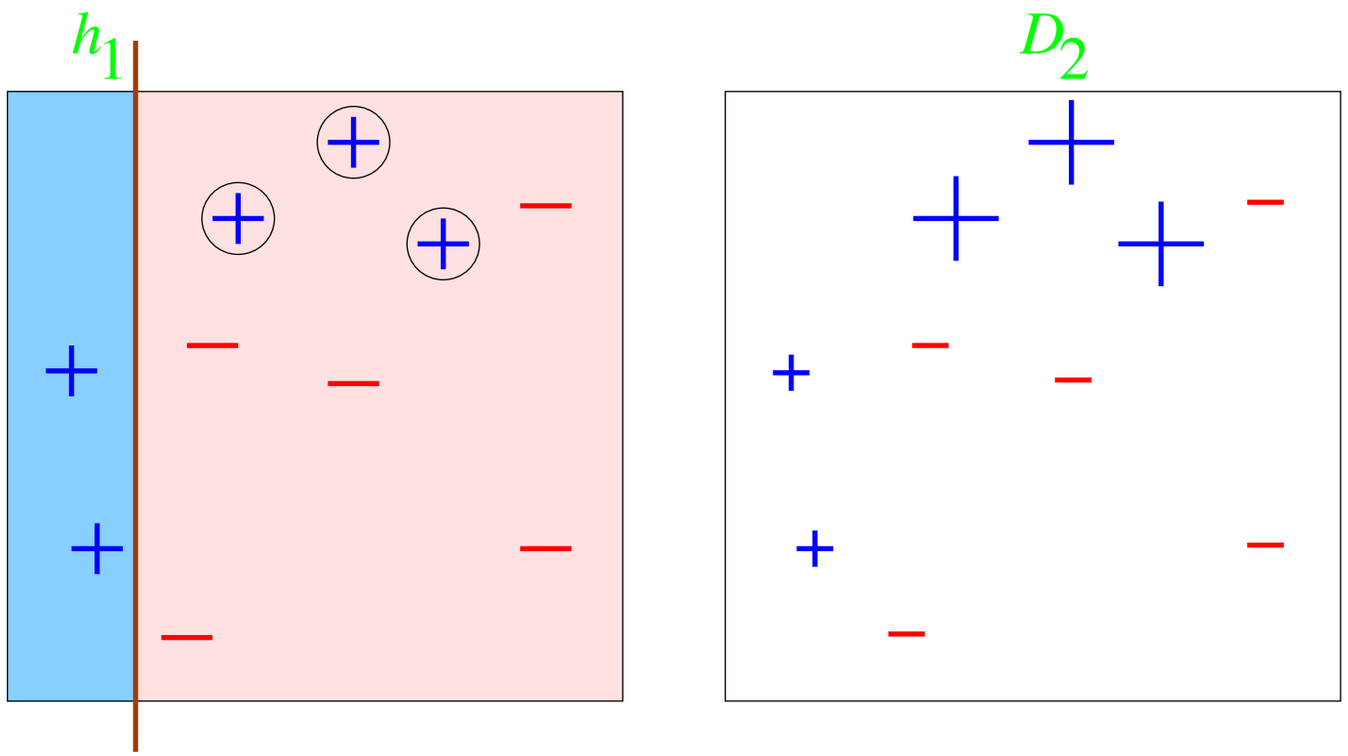
Toy Example



weak classifiers = vertical or horizontal half-planes

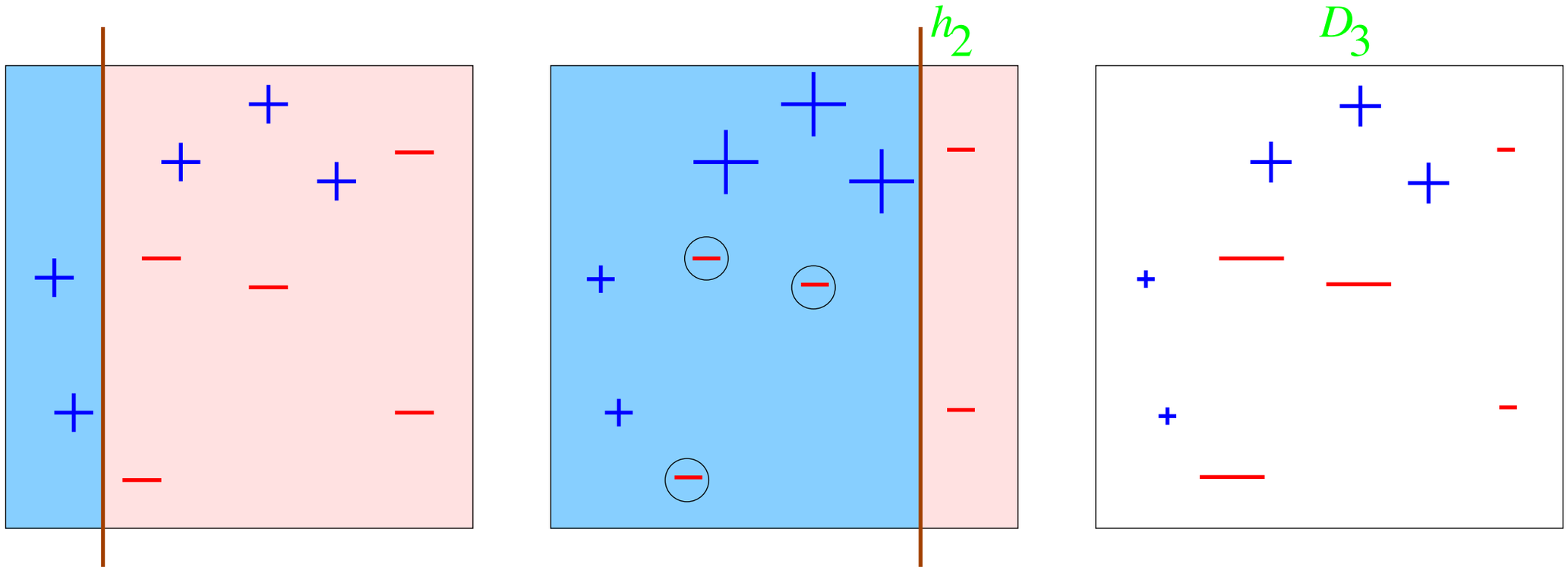
Slide taken from Schapire's NIPS 2007 tutorial
"Theory and Applications of Boosting"

Round 1



Slide taken from Schapire's NIPS 2007 tutorial
"Theory and Applications of Boosting"

Round 2

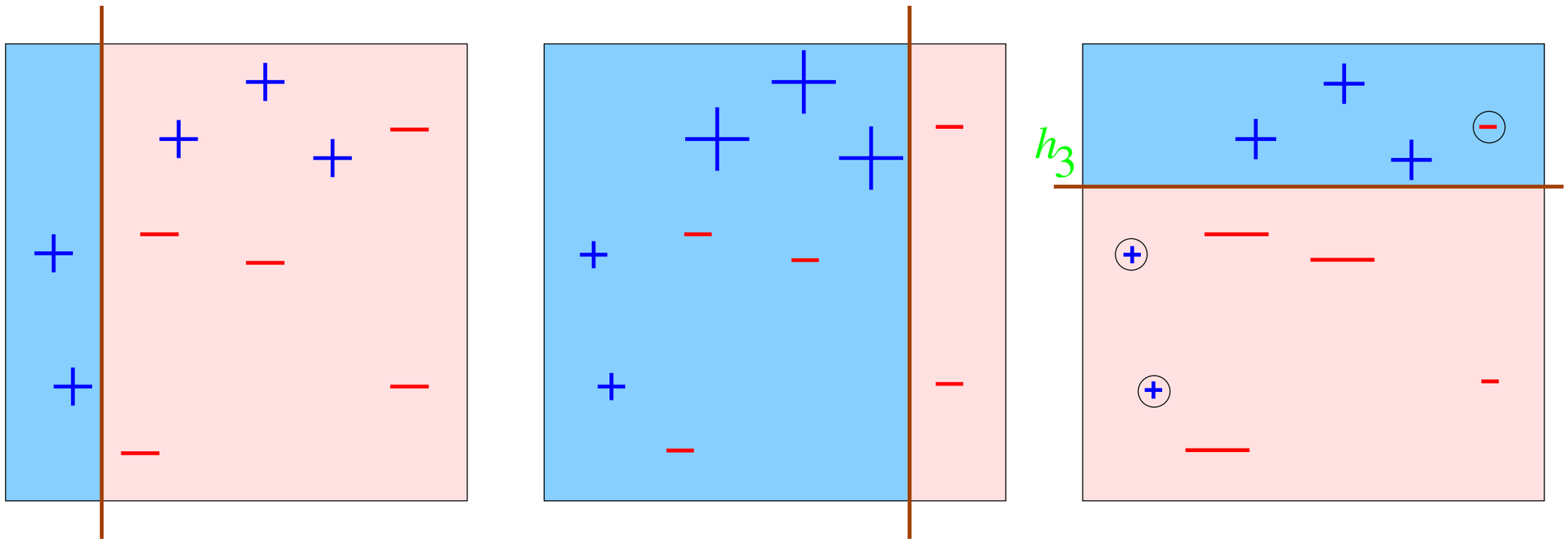


$\varepsilon_2 = 0.21$

$\alpha_2 = 0.65$

Slide taken from Schapire's NIPS 2007 tutorial
"Theory and Applications of Boosting"

Round 3

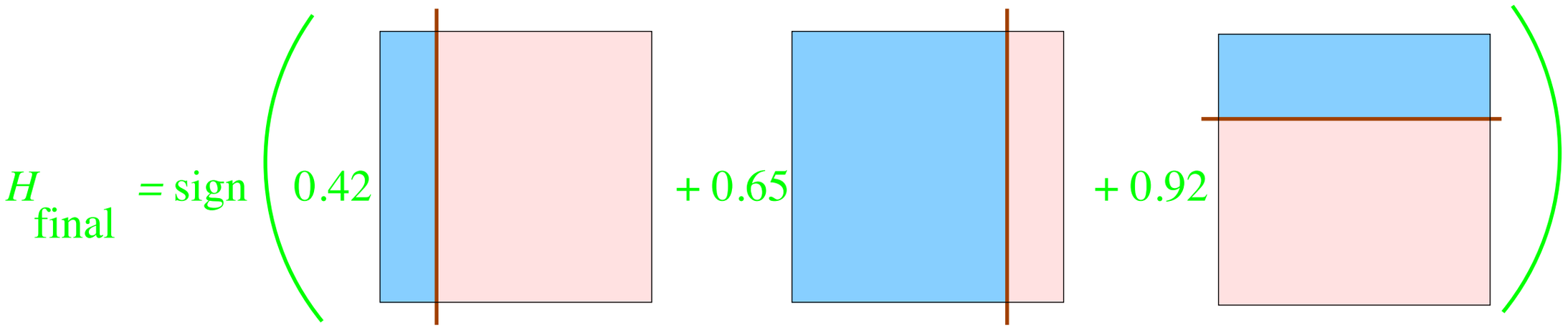


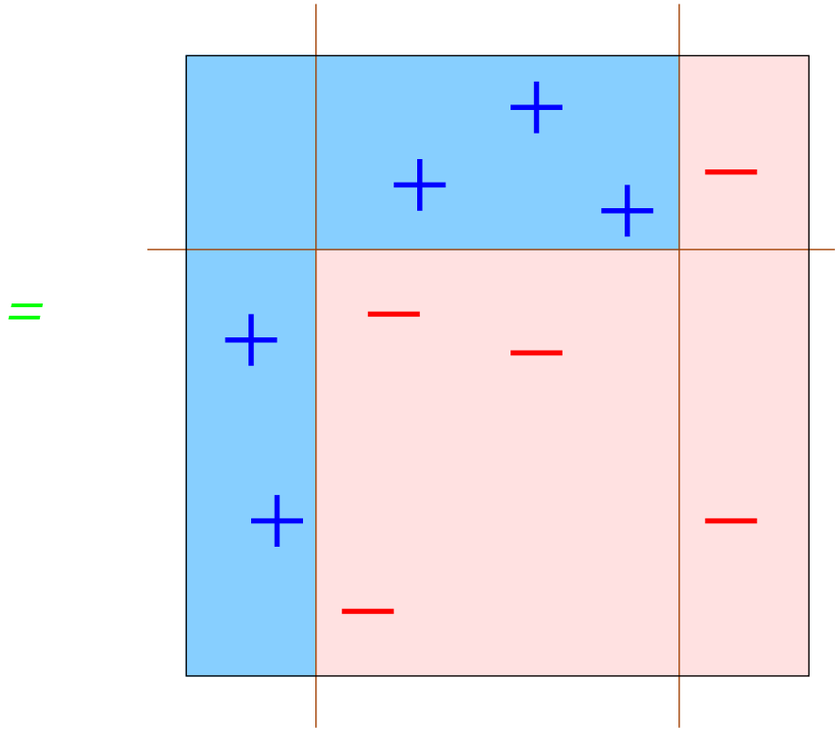
$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

Slide taken from Schapire's NIPS 2007 tutorial
"Theory and Applications of Boosting"

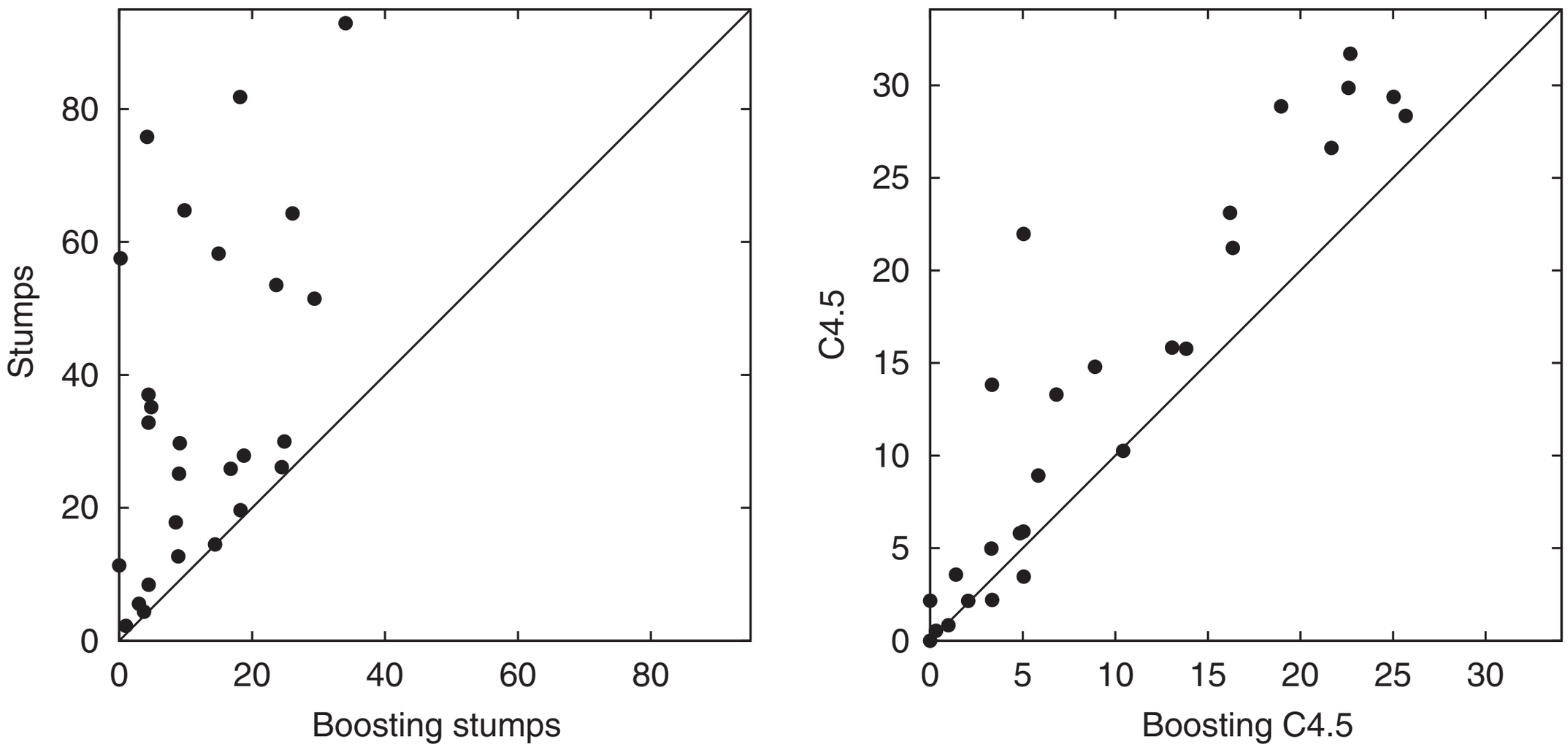
Final Classifier





Slide taken from Schapire's NIPS 2007 tutorial
"Theory and Applications of Boosting"

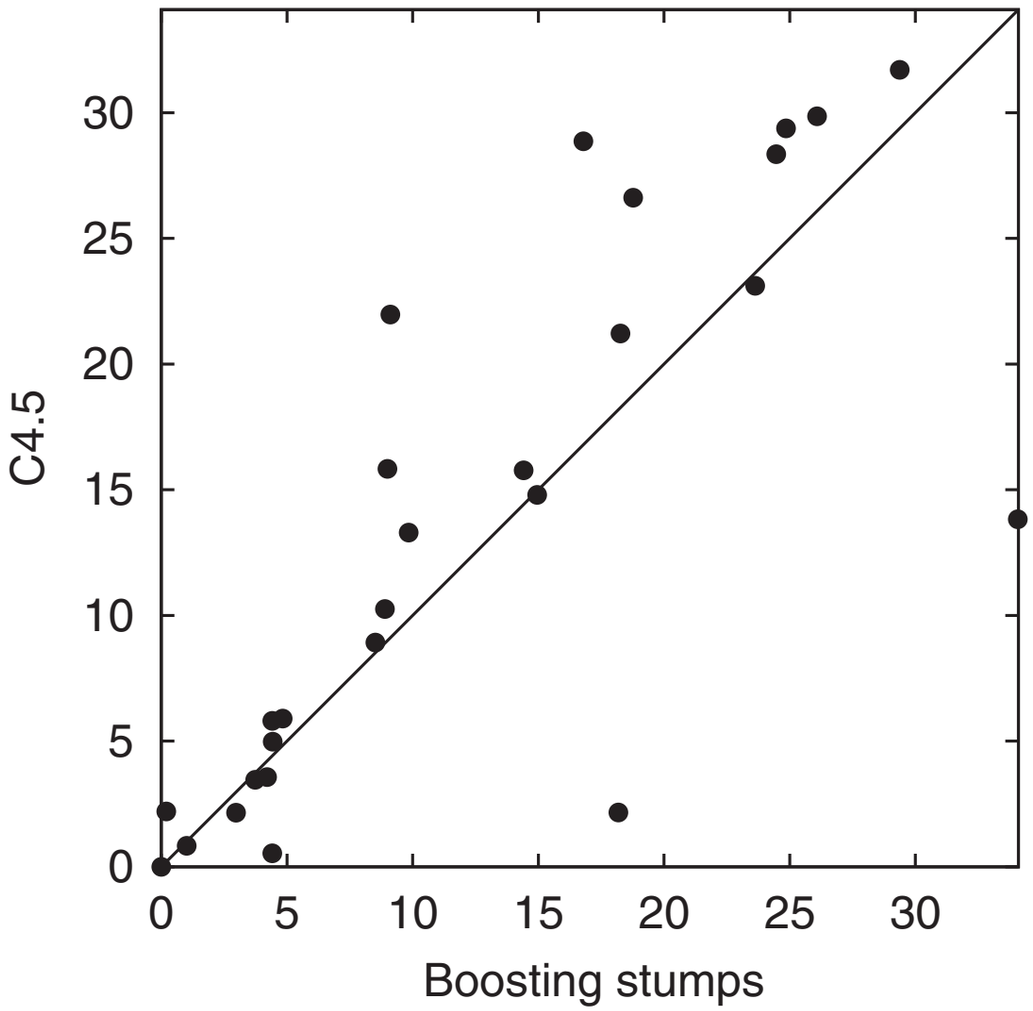
Boosting weak (and not-so-weak) learners

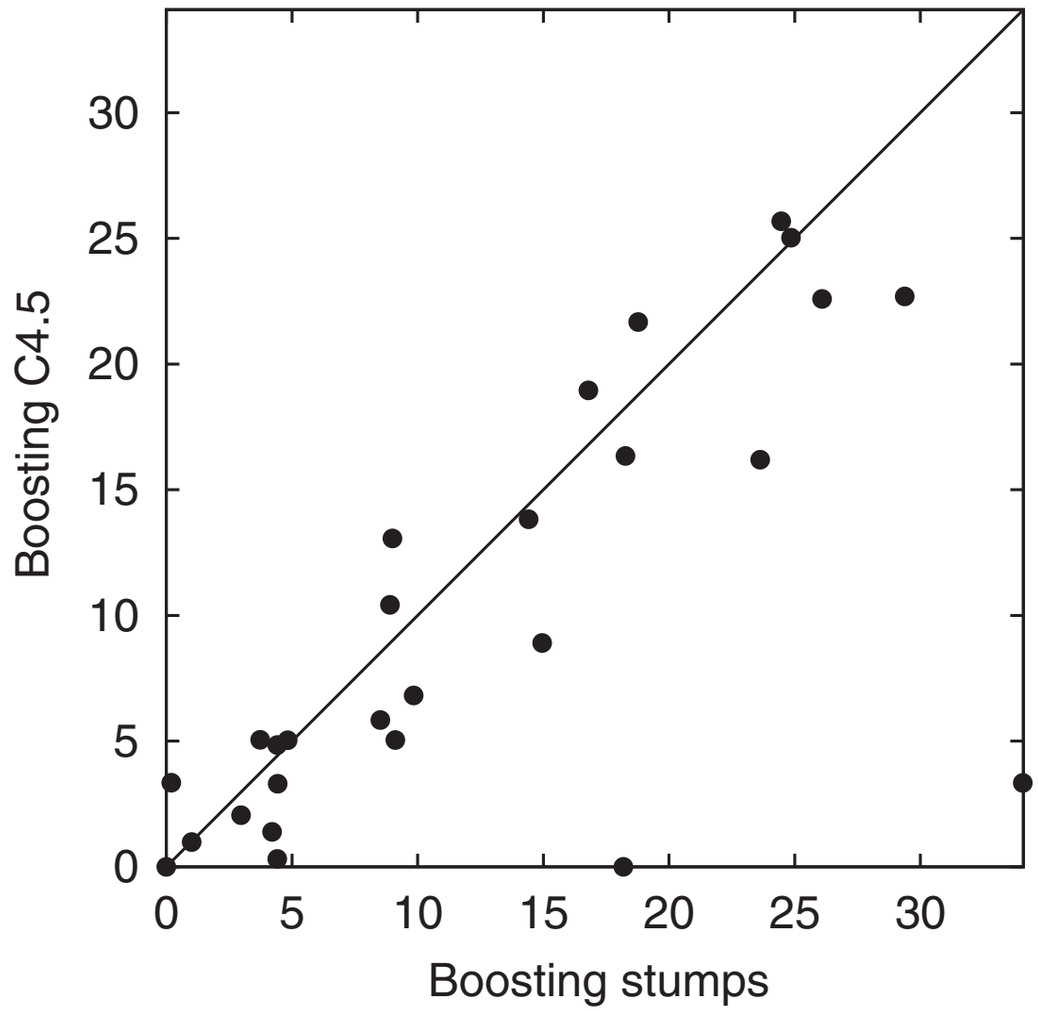


Each point shows the test error rate of the two competing algorithms on one of 27 benchmark learning problems. The *x*-coordinate of each point gives the percent test error using boosting, and the *y*-coordinate gives the error rate without boosting when using decision stumps (left plot) or C4.5 (right plot). All error rates have been averaged over multiple runs.

Figure taken from Schapire and Freund
(Boosting: Foundations and Algorithms, 2012)

More boosting comparisons





Each point shows the test error rate of the two competing algorithms on one of 27 benchmark learning problems.

Figure taken from Schapire and Freund
(Boosting: Foundations and Algorithms, 2012)

Training error of AdaBoost

Does AdaBoost provably work?

Let's first consider the training error of its final classifier ***H***

Theorem

We call γ_t the "*edge*" of h_t with respect to distribution D_t
(higher is better)

For each round ***t***, express ε_t as $\varepsilon_t = \frac{1}{2} - \gamma_t$

Then

$$\frac{1}{n}\sum_{i=1}^{n} 1\left[H(x_i) \neq y_i\right] \leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right)$$

Training Error

If all edges satisfy $\gamma_t \geq \gamma$ for some strictly positive γ
(if each blockhead does strictly better than chance…)
then training error decays exponentially fast as $e^{-2\gamma^2 T}$

Test Error of AdaBoost

As a sanity check, low training error is good…

…but what we *really* care about is getting low test error!

How large should the number of rounds T be for AdaBoost to get low test error?

A first intuition: If the weak learner selects hypotheses from some class $\mathcal{C}_{\text{weak}}$, and if the strong learner outputs a weighted majority vote among T hypotheses from $\mathcal{C}_{\text{weak}}$, then as T increases, the weighted majority vote is more complicated and has a higher potential to overfit.

Occam's razor: All things being equal (given two hypotheses with equal training error), select the simpler one.

What we expect from Occam's razor

The rough intuition suggests that we should definitely stop training as soon as the training error hits zero (or maybe even earlier…)

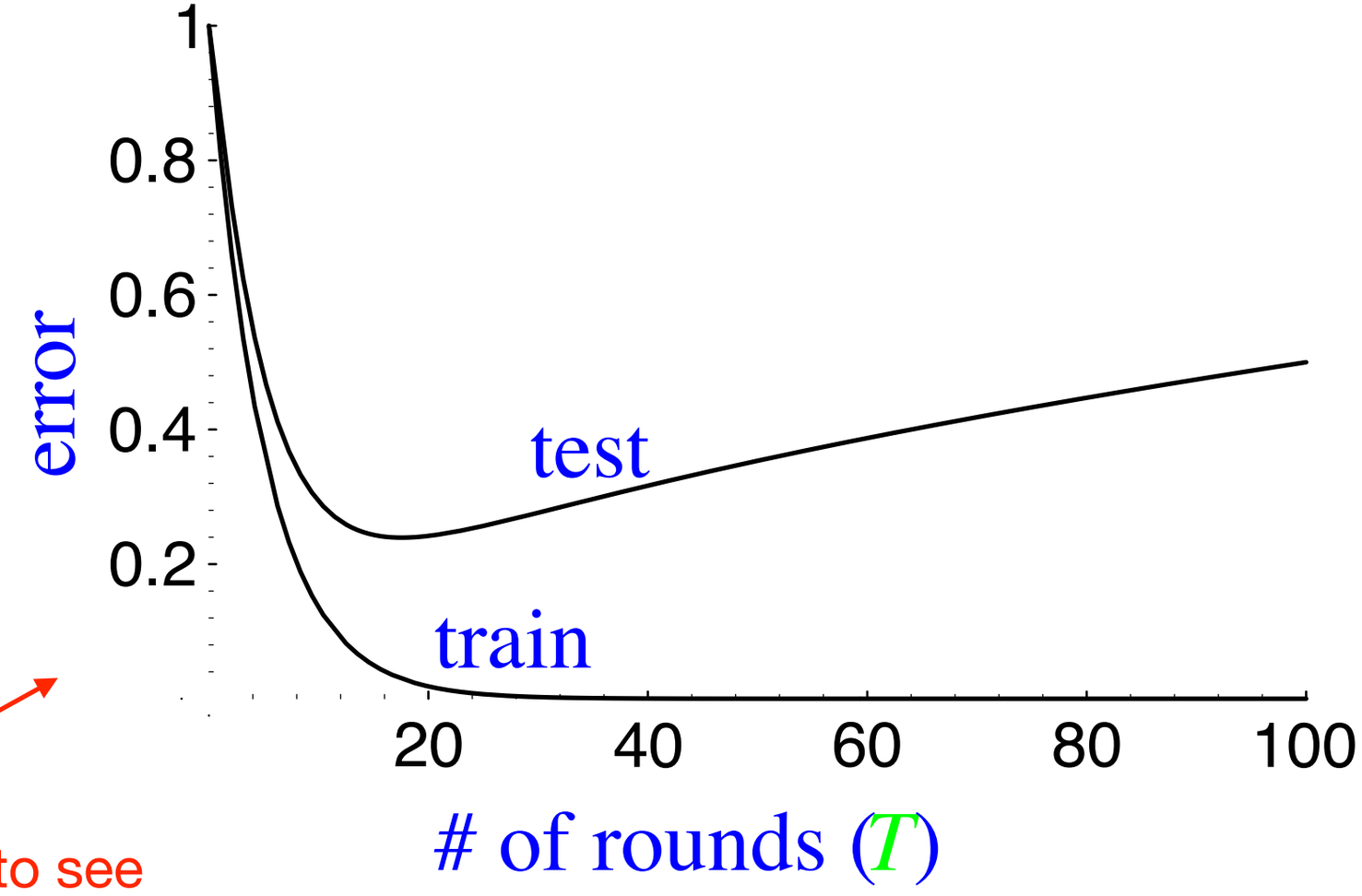
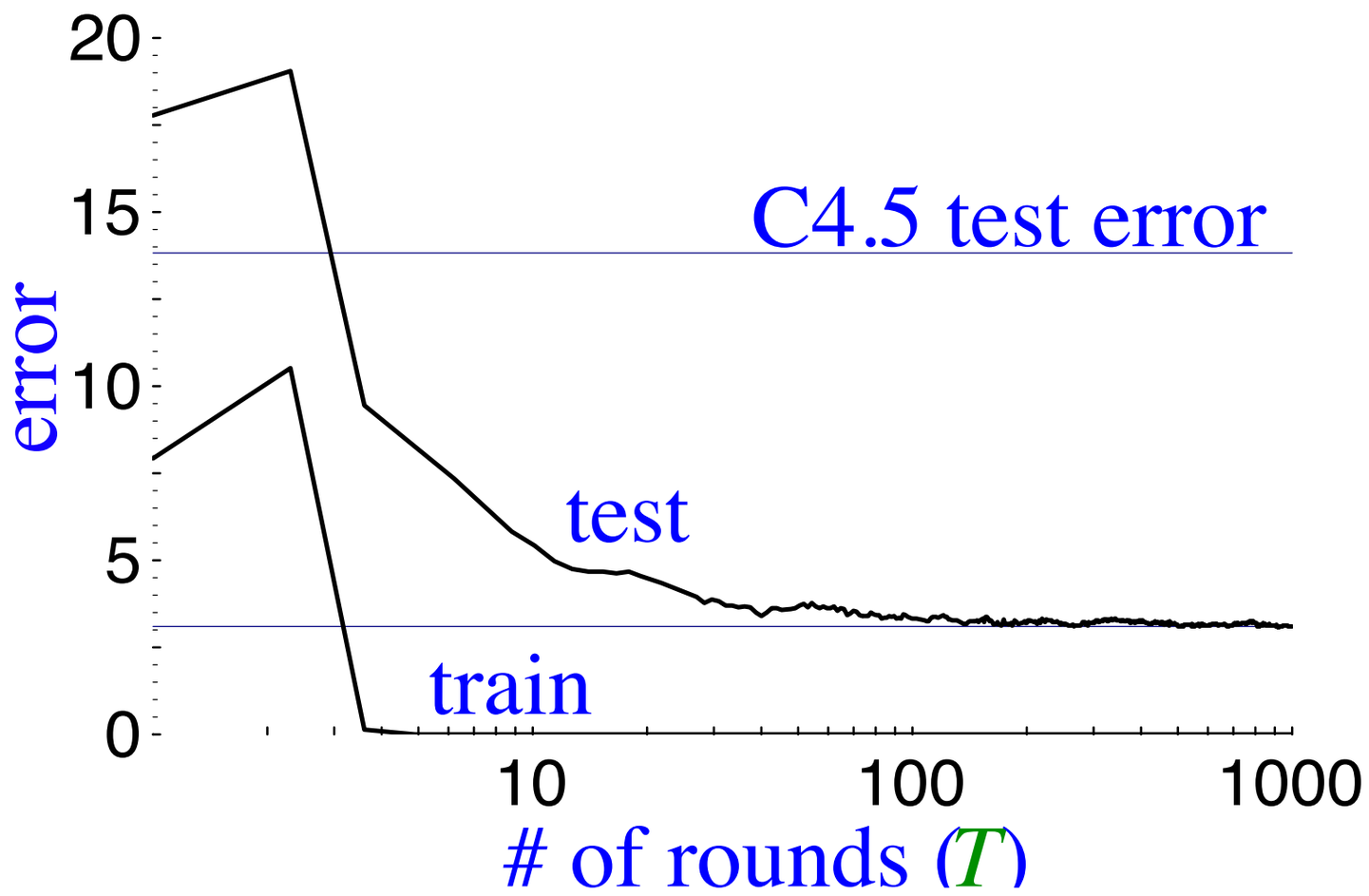


Figure we ***expect*** to see

Figure taken from Schapire's NIPS 2007 tutorial
"Theory and Applications of Boosting"

What actually often happens in practice

But in many cases, even if we train for a very large number of rounds (far beyond when the training error became exactly zero and stayed there!), the test error continues dropping



(boosting C4.5 on "letter" dataset)

Figure taken from Schapire's NIPS 2007 tutorial "Theory and Applications of Boosting"

What also can happen on hard problems

Yet, on some harder problems, we do get the familiar test error “elbow" curve

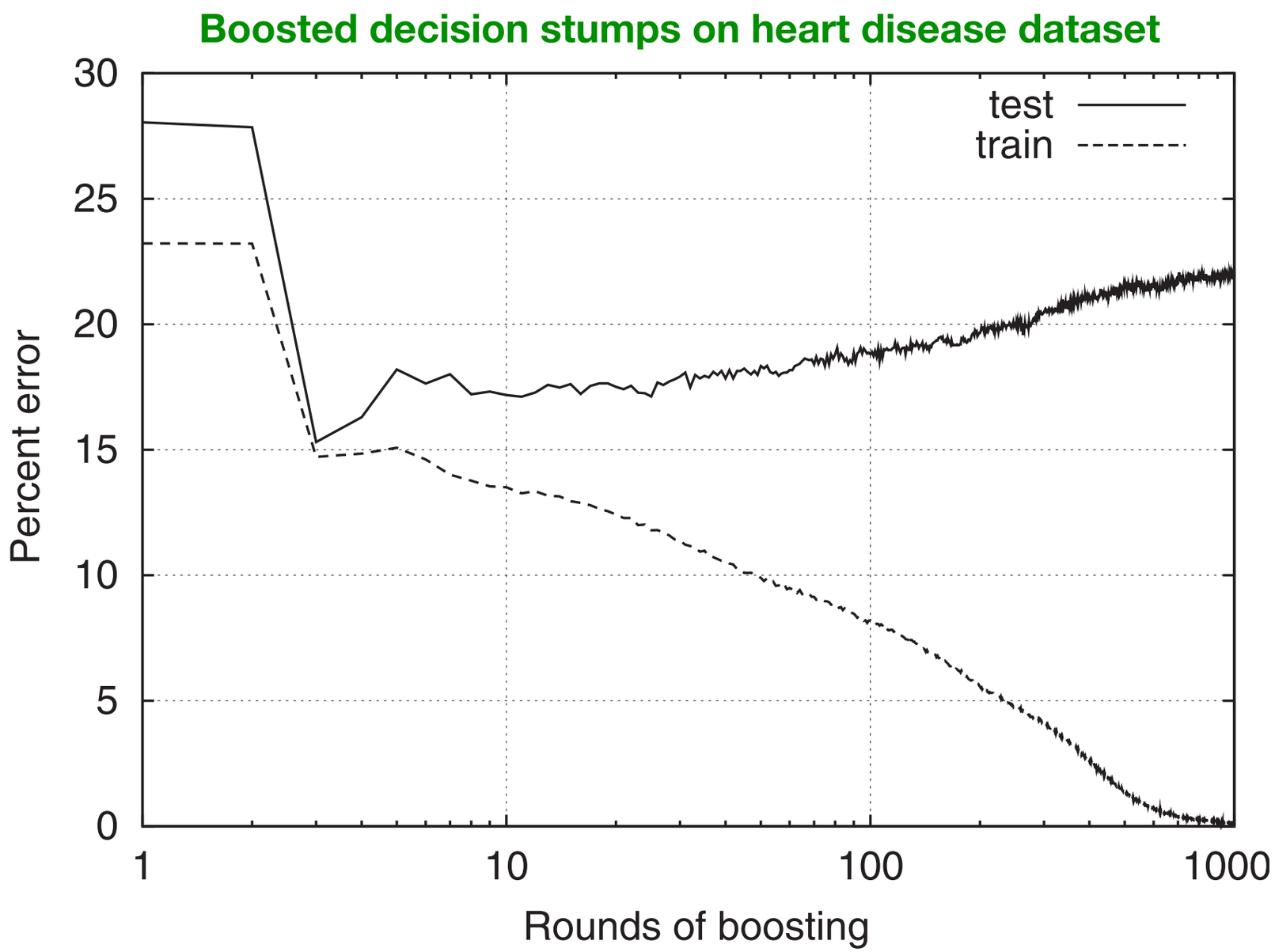


Figure taken from Schapire and Freund
(Boosting: Foundations and Algorithms, 2012)

Understanding the test error

So, what's going on here?

To understand this phenomenon, first define normalized weight $a_t = \dfrac{\alpha_t}{\sum_{s=1}^{T} \alpha_s}$

Now, given a weighted majority vote hypothesis H, we define the *margin* of H on any labeled example (x, y) as

$$[\text{total weight on correct hypotheses}] - [\text{total weight on incorrect hypotheses}]$$

$$\sum_{t=1}^{T} a_t 1\left[h_t(x) = y\right] - \sum_{t=1}^{T} a_t 1\left[h_t(x) \neq y\right]$$

Understanding the test error

So, what's going on here?

To understand this phenomenon, first define normalized weight $a_t = \dfrac{\alpha_t}{\sum_{s=1}^{T} \alpha_s}$

Now, given a weighted majority vote hypothesis H, we define the *margin* of H on any labeled example (x, y) as

$$[\text{total weight on correct hypotheses}] - [\text{total weight on incorrect hypotheses}]$$

$$\sum_{t=1}^{T} a_t 1\left[h_t(x) = y\right] - \sum_{t=1}^{T} a_t 1\left[h_t(x) \neq y\right]$$

Understanding the test error

So, what's going on here?

To understand this phenomenon, first define normalized weight $a_t = \dfrac{\alpha_t}{\sum_{s=1}^{T} \alpha_s}$

Now, given a weighted majority vote hypothesis H, we define the *margin* of H on any labeled example (x, y) as

$$[\text{total weight on correct hypotheses}] - [\text{total weight on incorrect hypotheses}]$$

$$\sum_{t=1}^{T} a_t 1\left[h_t(x) = y\right] - \sum_{t=1}^{T} a_t 1\left[h_t(x) \neq y\right]$$

If on a fixed example (x, y), 80% of the weight is on correctly predicting rules, then the margin is 0.8 - 0.2 = 0.6.

Understanding the test error

So, what's going on here?

To understand this phenomenon, first define normalized weight $a_t = \dfrac{\alpha_t}{\sum_{s=1}^{T} \alpha_s}$

Now, given a weighted majority vote hypothesis H, we define the *margin* of H on any labeled example (x, y) as

$$[\text{total weight on correct hypotheses}] - [\text{total weight on incorrect hypotheses}]$$

$$\sum_{t=1}^{T} a_t 1\left[h_t(x) = y\right] - \sum_{t=1}^{T} a_t 1\left[h_t(x) \neq y\right]$$

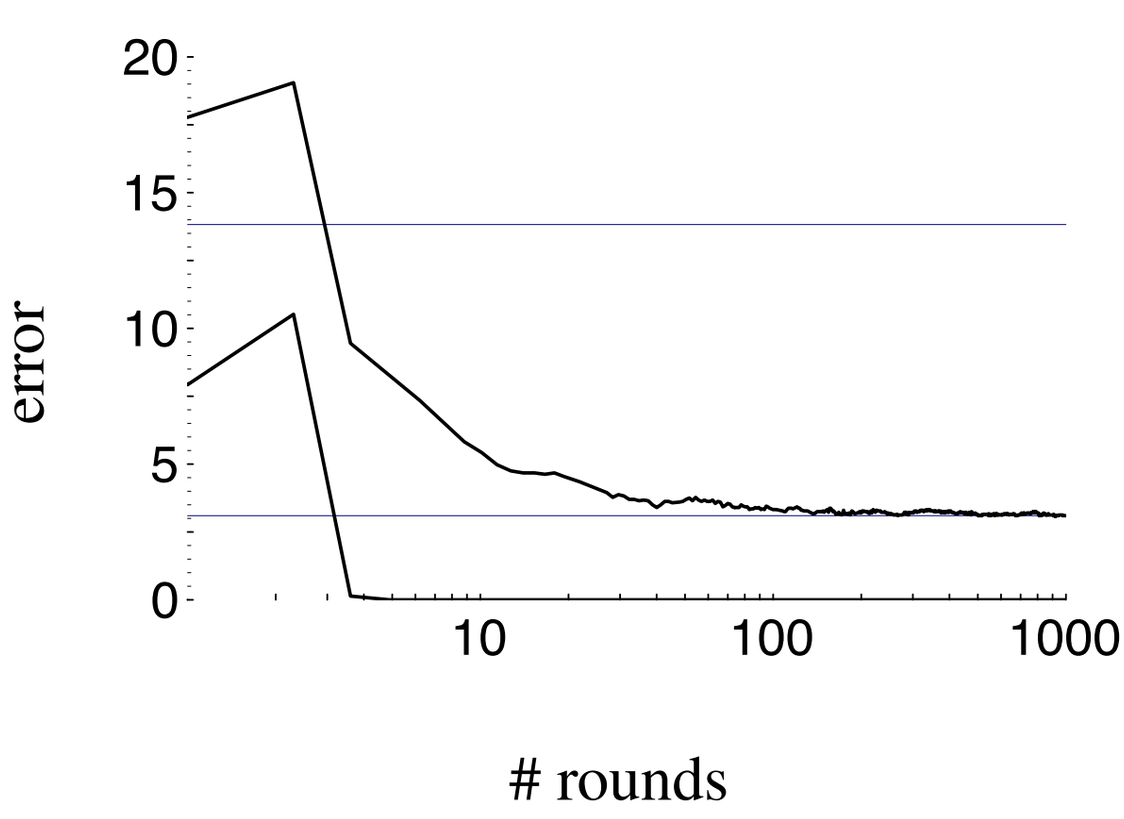
If on a fixed example (x, y), 80% of the weight is on correctly predicting rules, then the margin is 0.8 - 0.2 = 0.6.

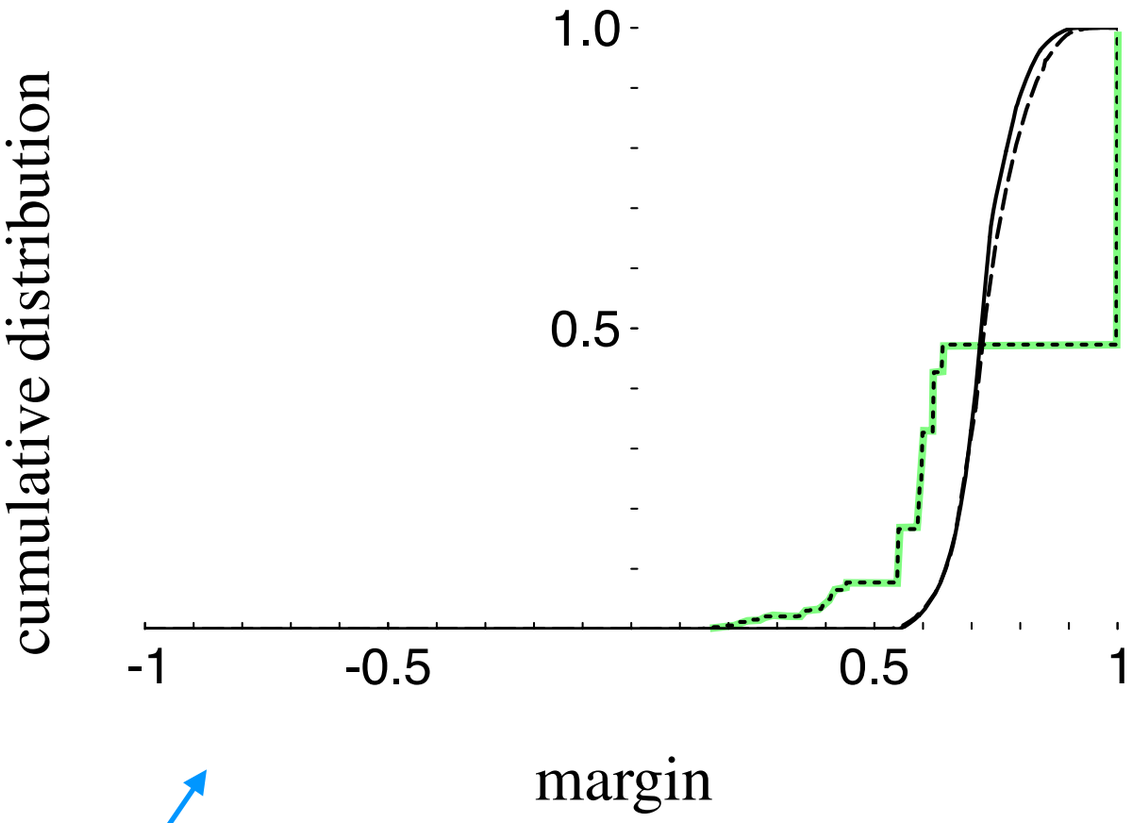
We can then study the *empirical distribution of the margin* over the training sample.

Empirical margin distribution

Another look at the example where test error kept decreasing:

After 10 rounds, all examples have margin ≥ 0.14



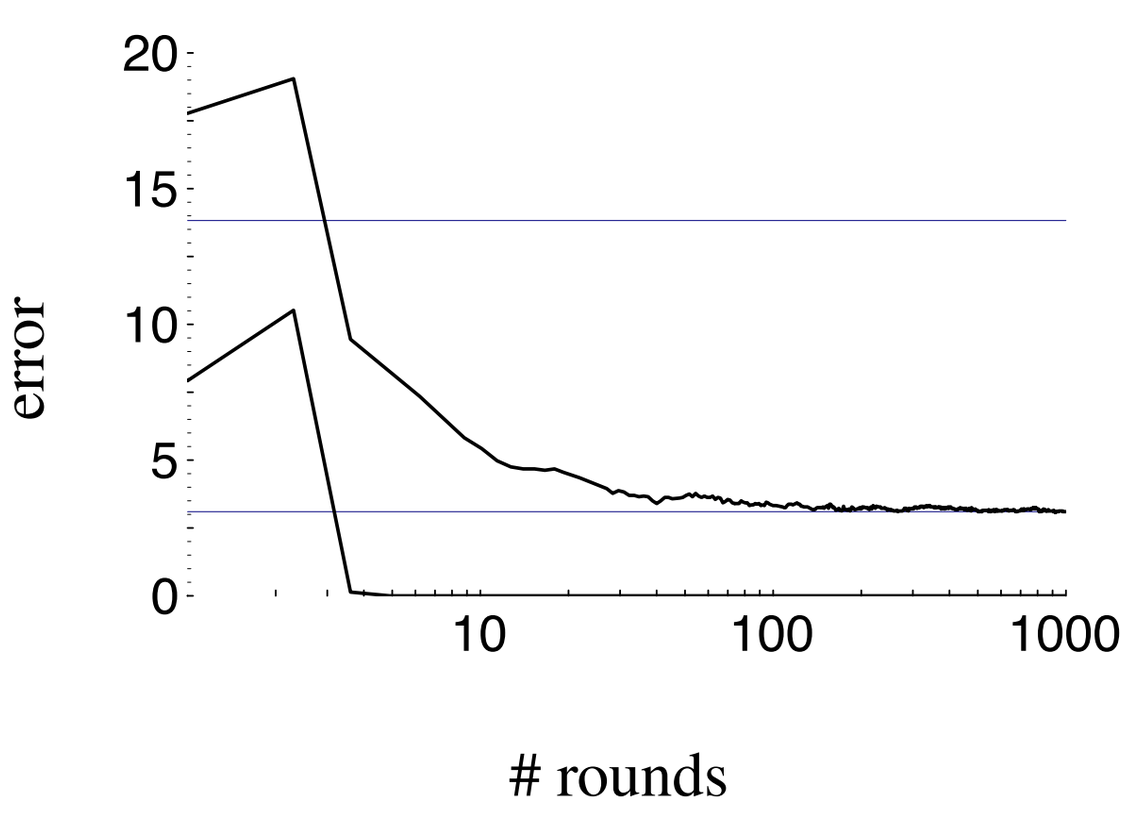


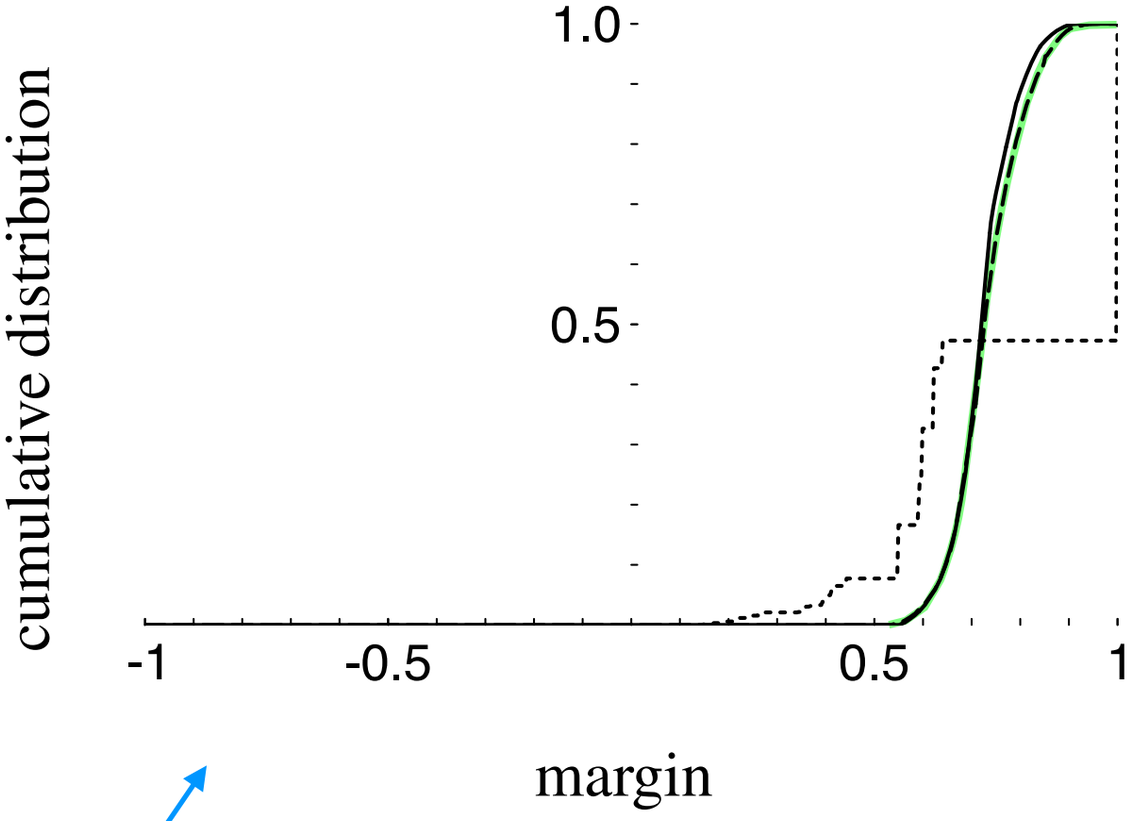
The cumulative distribution of margins of the training examples after 5, 100 and 1000 iterations, indicated by short-dashed, long-dashed, and solid curves, respectively.

Empirical margin distribution

Another look at the example where test error kept decreasing:

After 100 rounds, all examples have margin ≥ 0.52



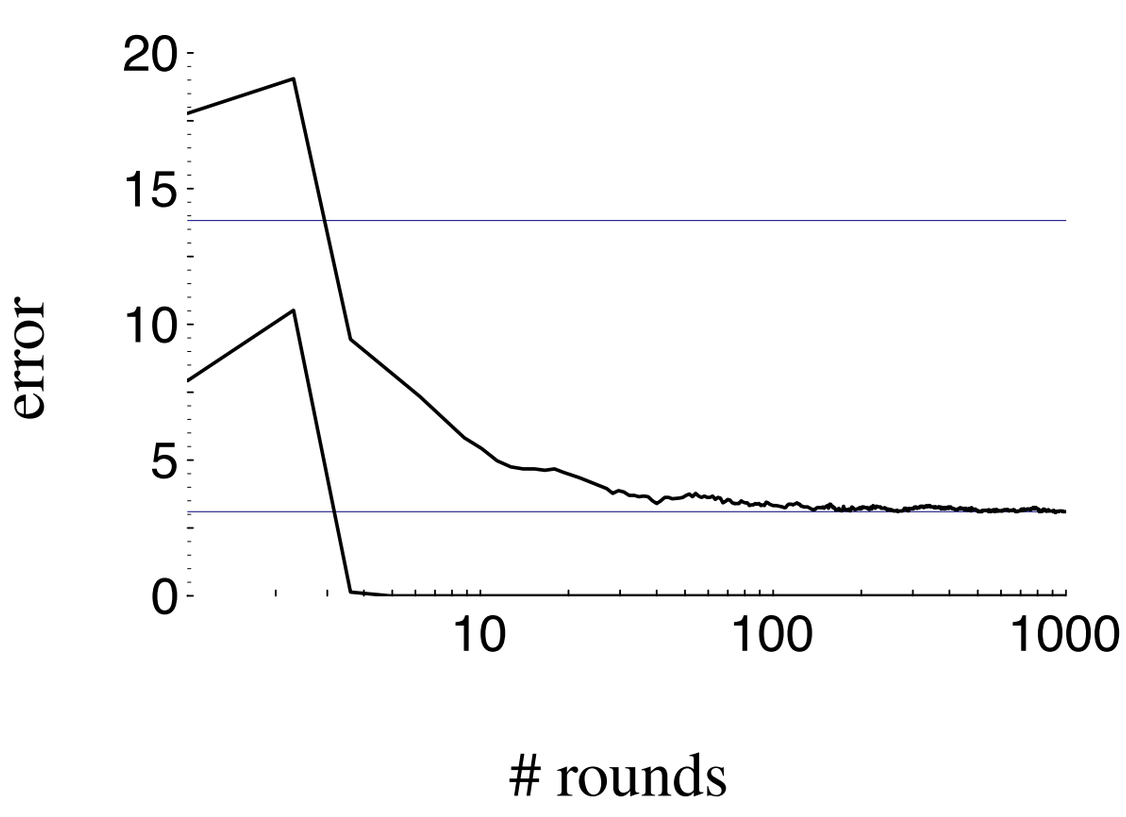


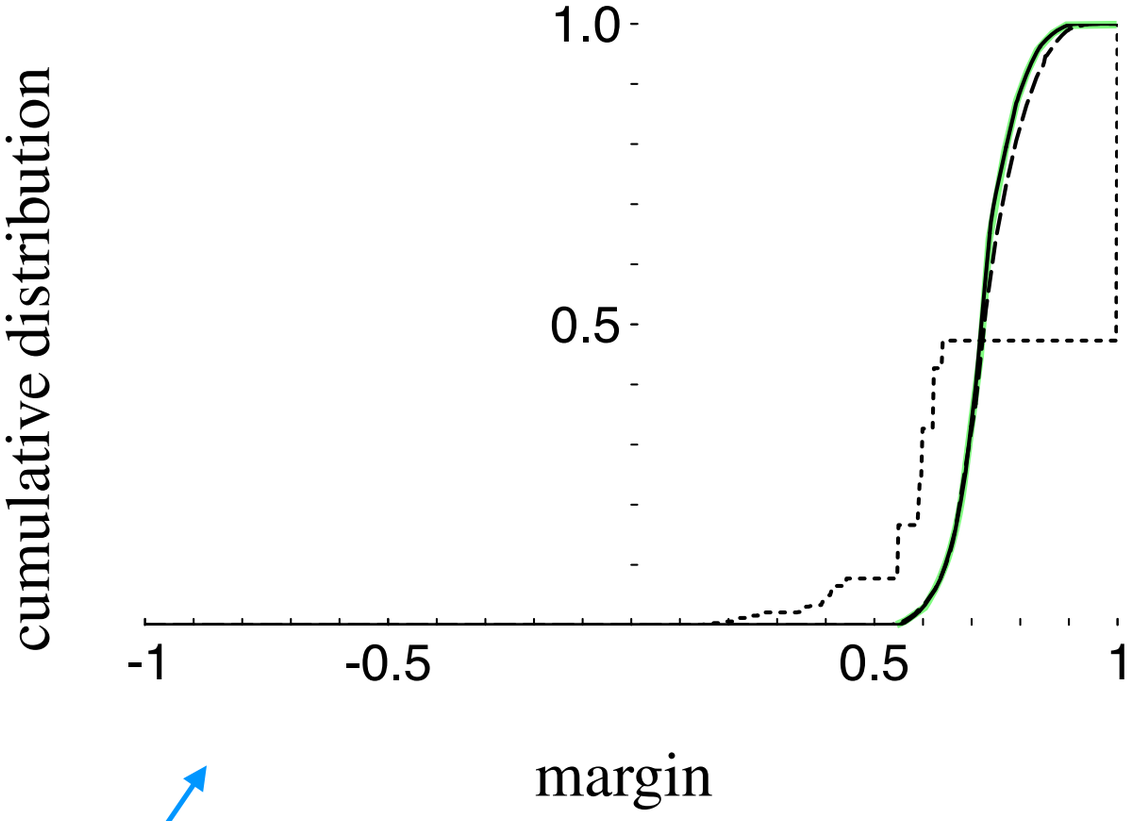
The cumulative distribution of margins of the training examples after 5, 100 and 1000 iterations, indicated by short-dashed, long-dashed, and solid curves, respectively.

Empirical margin distribution

Another look at the example where test error kept decreasing:

After 1000 rounds, all examples have margin ≥ 0.55





The cumulative distribution of margins of the training examples after 5, 100 and 1000 iterations, indicated by short-dashed, long-dashed, and solid curves, respectively.

The margins explanation for AdaBoost

OK, after a large number of rounds, AdaBoost often yields a weighted majority that has a large margin for most examples. Why does that help?

Think about a massive election involving a billion voters where:

- 75% of the population votes for candidate A
- 25% of the population votes for candidate B

Could we get the same winner using only a small random sample of the voters?

The margins explanation for AdaBoost

OK, after a large number of rounds, AdaBoost often yields a weighted majority that has a large margin for most examples. Why does that help?

Think about a massive election involving a billion voters where:

- 75% of the population votes for candidate A
- 25% of the population votes for candidate B

Could we get the same winner using only a small random sample of the voters?

Yes. Think of a vote for A as 1 and a vote for B as -1, so the mean vote is equal to 0.75 - 0.25 = 0.5.

We just need our “subsample election" to have a positive sample mean.

The margins explanation for AdaBoost

OK, after a large number of rounds, AdaBoost often yields a weighted majority that has a large margin for most examples. Why does that help?

Think about a massive election involving a billion voters where:

- 75% of the population votes for candidate A
- 25% of the population votes for candidate B

Could we get the same winner using only a small random sample of the voters?

Yes. Think of a vote for A as 1 and a vote for B as -1, so the mean vote is equal to 0.75 - 0.25 = 0.5.

We just need our “subsample election" to have a positive sample mean. With k randomly selected voters, from Hoeffding's inequality the sample mean will be at least $0.5 - O(1/\sqrt{k})$ with high probability.

So, take $k \approx 1/0.5^2 = 4$ (this depends only on the margin, not on T (!))

In general, for margin θ, we use $k = O(1/\theta^2)$

A margin-based bound

V is a measure of complexity of the weak learner's hypothesis class
(example we will cover later: *V* = VC dimension)

With high probability, for all $\theta > 0$,

$$\text{Risk} \leq [\text{fraction of points where margin} \leq \theta] + O\left(\frac{1}{\theta}\sqrt{\frac{V}{n}}\right)$$

As long as $\theta < \gamma$ this decays exponentially quickly to 0
as *T* increases (just like the training error)

Take-home message: Having a weighted majority vote among a very large number of hypotheses (large T) does not hurt at all, and can in fact help, if the empirical margin distribution gets better (more training points have large margin) as *T* grows.